### **Assignment: Implementing RDBMS Schema in Apache Hive**

**Introduction**

Apache Hive was created by Facebook in 2021 for non-programmers familiar with petabytes of data. To this day it is a powerful batch processing system that is built on Hadoop MapReduce utilizing the Hadoop Distributed File Systems to handle enormous amounts of data. Great for data warehousing tasks, reporting, and analytics. It supports a SQL interface which is familiar to many, making it easier to integrate. There is also the ability to create complex queries such as aggregations for analytics of huge data sets. This system gives a company the ability to provide insights to large datasets in a timely manner that can support client applications with minimal risk, largely due to the HDFS architecture.

By implementing our RDBMS in Hive, we hope to measure performance gains or losses to either affirm that moving to Hive is a change that a company with large sets of data should consider or not. With the ability to quickly query large sets of data through batch processing, we would like to also consider being able to do more analytics supporting front facing client interfaces. In a time where everyone expects immediate results, Hive gives clients the tool for insight into large datasets without the data becoming old and stale due to processing time. There is also a reliability aspect important when in business via the replication process provided by HDFS.

The following points reflect the considerations that must be taken to implement an RDBMS schema into Hive as well as what our team will focus on understanding:

* Reliability – HDFS replication
* Speed - HDFS multi nodes
* Large Processing - HDFS multi nodes
* Optimization – HDFS partitions and constraints
* Data warehousing
* Large Scale ETL

**Data Manipulation and Querying**

Below are eight scenarios our team is trying to understand via HiveQL queries:

Q1c) Item Sales Order by revenue center

Q2c) Store Sales Order by revenue center

Q1s) High-performing Item Category per revenue center in terms of revenue

Q2s) Under performing Item Category per revenue center in terms of revenue

Q1j) Top performing regions per capita

Q2j) Worst performing regions per capita

Q1l) Check demand levels (quantity) of specified items in each revenue center.

Q2l) calculating profit margin for each store after accounting for the cost.

**Performance Considerations**

Hive has many advantages over other architectures. Its design is meant for large sets of data as it is built on HDFS and processes queries quickly due to its batch processing capabilities. Hive also contains a SQL interface that is an advantage for those who have historically worked with the SQL syntax. When building this architecture, our team relied on their familiarity with the SQL language to create and modify queries appropriately. However, a downside of using Hive is that it does not support complex SQL features – like full joins - which posed some challenges early in the design process. Our team also had to weigh the importance of real-time querying for our specific use-case since Hive is not able to do this efficiently. Thus, the queries our team chose were created with these limitations in mind. Although this was a challenge, other aspects of Hive were beneficial to the performance of our dataset. Hive’s underlying HDFS layer promotes the reliability of the data. For example, if a node containing data a client is trying to query is down, another node containing a backup copy of this data can provide that to the client, making this an attractive feature for clients looking to have data available at all times.

Hive’s processing power is one of its biggest advantages against other architectures like RDBMS and Cassandra. In contrast to its fast querying, there are some other significantly less desirable differences between these architecture designs. Most notably is Hive’s inability to understand complex joins which are very prominent in relational databases. Scalability is also a feature that is not as strong in Hive in comparison to Cassandra. Scalability plays a big role in scenarios where large amounts of data may need to be stored and scaled seamlessly. This can be accomplished with relative ease in Cassandra, whereas Hive may see a significant drop in performance when data size becomes larger.

**Challenges and Learnings**

Many of the challenges our team faced during this implementation were related to the syntax of the querying language. Despite being able to leverage SQL, the team had to restructure the queries to be compatible with Hive’s syntax. Hive also has limited data types, which became a challenge when creating our tables due to our table containing monetary data types that are not readily available in Hive. In terms of the HDFS, this system has its own way of accessing, processing, and retrieving data back to the client compared to a relational database. The HDFS schema adds an additional layer of complexity to the design due to its multi-node structure.

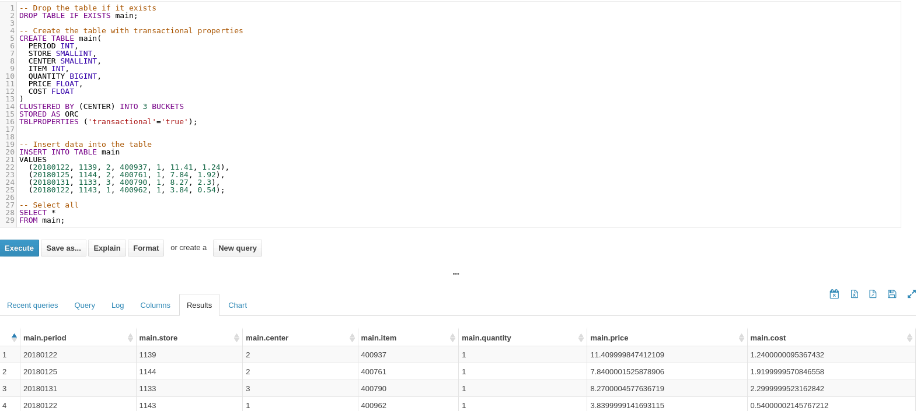
Given the similarities and differences between all architectures, choosing an architectural design for a dataset depends on the use case and project requirements. When real-time, quick data access is needed, a Cassandra architecture is more suitable. On the other hand, if these factors are not of much importance and other factors – such as cost – take precedence, Hive should be considered; it is still suitable for large amounts of data but will minimize operational costs if data latency is not an issue.

**Conclusion**

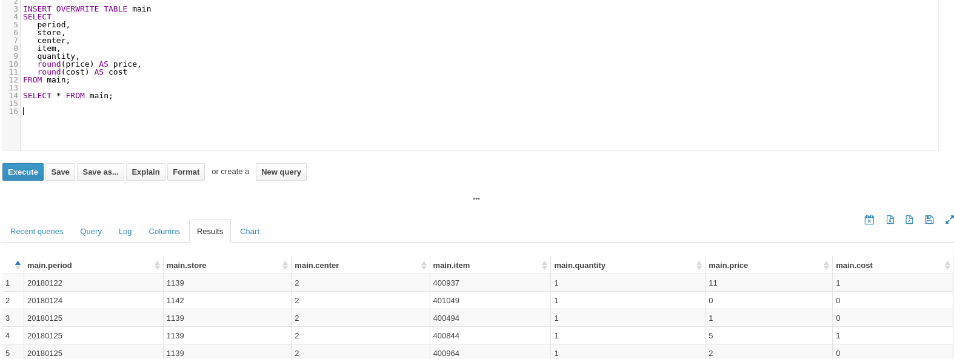
Using a Hive framework benefited our data by increasing reliability, speed, and optimization. When designing our tables, we determined what the best partition keys would be. This is because Hive’s HDFS layer uses the partition key to segment data in different tables for optimal and faster performance. The framework also contains multiple nodes in which it can store the data in, improving the speed of processing queries especially when using a large dataset. Additionally, by making use of its HDFS replication capabilities, data is stored in a primary and secondary node to avoid its availability being impacted by a node failure.

These three aspects – optimization, speed, and reliability – are factors that must be considered when building big data applications and data warehouses in the real world. The importance that each of these would have on a specific dataset is dependent on the business objectives and what the organization is trying to understand from the data, this is why it is important to assess each of these criteria and weigh them against the cost to implement and maintain the solution. In addition to Hive, other architecture designs can be taken into consideration as they may be a better fit for an organization’s needs. For this project, we housed our data in both Hive and Cassandra, each having its own impact, advantages, and disadvantages for our use case. Given our business needs to explore the data and build detailed queries, the Hive architecture is better suited for our dataset. Cassandra’s advantages over Hive, such as its ability to scale easily, is not of concern of ours due to real-time analysis not being necessary to answer the queries we proposed. Therefore, we can make more use of Hive’s features in comparison to Cassandra.

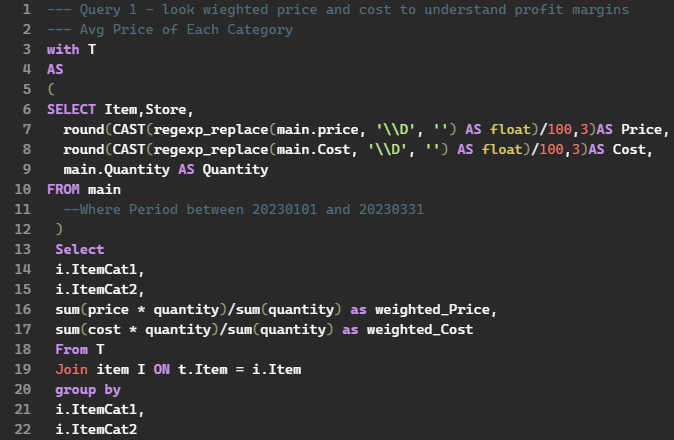
* HiveQL
  + Table Schema:



* Insert Overwrite (optional: update workaround)



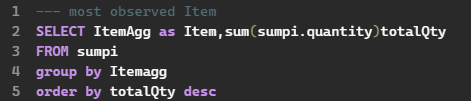
* Query1 w/ snapshot

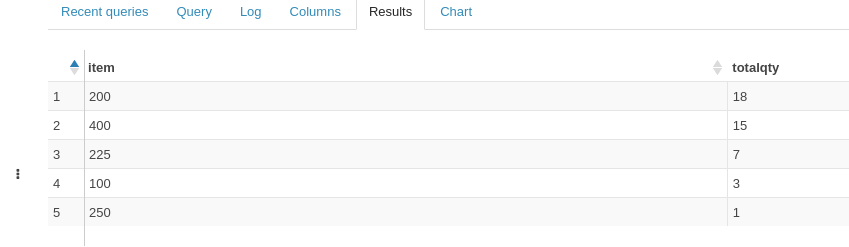


A screenshot of a computer

Description automatically generated

* Query2 w/ snapshot





* Query3 w/ snapshot

A screenshot of a computer program

Description automatically generated

